Order-Sorted Unification with Regular Expression Sorts
(Work in Progress)

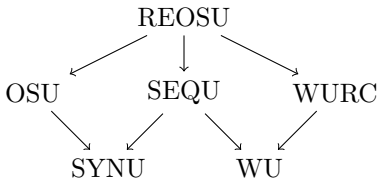
Temur Kutsia[1]⋆ and Mircea Marin[2]⋆⋆

[1] Research Institute for Symbolic Computation
Johannes Kepler University Linz, Austria
[2] Graduate School of Systems and Information Engineering
University of Tsukuba, Japan

Abstract. We extend first-order order-sorted unification by permitting regular expression sorts for variables and in the domains of function symbols. The set of basic sorts is finite. The corresponding unification problem is infinitary. We conjecture that this unification problem is decidable and give a complete unification procedure.

1 Introduction

In first-order order-sorted unification [15], the set of basic sort S is assumed to be partially ordered, variables are of basic sorts $\mathsf{s} \in S$ and function symbols have sorts of the form $\mathsf{w} \to \mathsf{s}$, where w is a finite word over S and $\mathsf{s} \in S$. In this paper, we require S to be finite and extend the framework by introducing regular expression sorts R over S, allowing variables to be of sorts R and function symbols to have sorts $\mathsf{R} \to \mathsf{s}$. Another extension is that overloading function symbols is allowed. Under some reasonable conditions imposed over the signature according to [7], our terms have a least sort. We call the obtained problem regular expression order-sorted unification (REOSU) and show that it is infinitary, conjecture that it is decidable and give a complete unification procedure. REOSU extends some known problems as it is shown on the diagram below, illustrating its relations with syntactic unification (SYNU [11]), word unification (WU [13]), order-sorted unification (OSU [15]), sequence unification (SEQU [10]), and word unification with regular constraints (WURC [13]):



⋆ Supported by JSPS Grant-in-Aid no. 20500025 for Scientific Research (C).
⋆⋆ Supported by the European Commission Framework 6 Programme for Integrated Infrastructures Initiatives under the project SCIEnce—Symbolic Computation Infrastructure for Europe (Contract No. 026133).

Following the arrows, from OSU one can obtain SYNU by restricting the sort hierarchy to be empty; SEQU problems without sequence variables (i.e., with individual variables only) constitute SYNU problems; on the other hand, WU is a special case of SEQU with constants, sequence variables, and only one flexible arity function symbol for concatenation; WU is also a special case of WURC where none of the variables are constrained; from REOSU we can get OSU (but with finitely many basic sort symbols only, because this is what REOSU considers) if instead of arbitrary regular sorts in function domains we allow only words over basic sorts, restrict variables to be of only basic sorts, and forbid function symbol overloading; SEQU can be obtained if we restrict REOSU with only one basic sort, say s, the variables that correspond to sequence variables in SEQU have sort s^*, individual variables are of sort s, and function symbols have the sort $\mathsf{s}^* \to \mathsf{s}$; finally, the WURC can be obtained from REOSU by the same restriction that gives WU from SEQU and, in addition, identifying the constants there to the corresponding sorts.

Order-sorted unification described in [12, 16] extends OSU from [15] in the way that is not compatible with REOSU.

In this paper we are dealing with REOSU in the empty theory (i.e., the syntactic case). As a future work, it would be interesting to see how equational OSU [9, 8] can be extended with regular expression sorts.

The paper is organized as follows: In Sect. 2 we give basic definitions and recall some known results. In Sect. 3 algorithms operating on sorts are given. Sect. 4 discusses decidability issues and describes a complete unification procedure. Sect. 5 concludes. Proofs can be found in the appendix.

For unification, we use the notation and terminology of [5]. For the notions related to sorted theories, we follow [7].

2 Preliminaries

Sorts. We consider a finite set $\mathcal{B}$ of basic sorts, partially ordered with the relation $\preceq$. Its elements are denoted with lower case letters in sans serif font. $\mathsf{s} \prec \mathsf{r}$ means $\mathsf{s} \preceq \mathsf{r}$ and $\mathsf{s} \neq \mathsf{r}$. Regular expression sorts (shortly, sorts) are regular expressions over $\mathcal{B}$, built in the usual way: $\mathsf{R} ::= \mathsf{s} \mid 1 \mid \mathsf{R}_1.\mathsf{R}_2 \mid \mathsf{R}_1{+}\mathsf{R}_2 \mid \mathsf{R}^*$.

The set of all regular expression sorts is denoted by $\mathcal{R}$. We use capital SANS SERIF font letters for them. They define the corresponding regular language in the standard way: $[\![\mathsf{s}]\!] = \{\mathsf{s}\}$, $[\![1]\!] = \{\epsilon\}$, $[\![\mathsf{R}_1.\mathsf{R}_2]\!] = \{(\tilde{s}_1, \tilde{s}_2) \mid \tilde{s}_1 \in [\![\mathsf{R}_1]\!], \tilde{s}_2 \in [\![\mathsf{R}_2]\!]\}$, $[\![\mathsf{R}_1{+}\mathsf{R}_2]\!] = [\![\mathsf{R}_1]\!] \cup [\![\mathsf{R}_2]\!]$, $[\![\mathsf{R}^*]\!] = [\![\mathsf{R}]\!]^*$, where ϵ stands for the empty word.

We extend the ordering $\preceq$ to words of basic sorts of equal lengths by $\mathsf{s}_1 \cdots \mathsf{s}_n \preceq \mathsf{r}_1 \cdots \mathsf{r}_n$ iff $\mathsf{s}_i \preceq \mathsf{r}_i$ for all $1 \leq i \leq n$. This ordering is extended to sets of words of basic sorts, defining $S_1 \preceq S_2$ iff for each $w_1 \in S_1$ there is $w_2 \in S_2$ such that $w_1 \preceq w_2$. Finally, we order the regular expression sorts with $\preceq$, defining $\mathsf{R}_1 \preceq \mathsf{R}_2$ iff $[\![\mathsf{R}_1]\!] \preceq [\![\mathsf{R}_2]\!]$. If $\mathsf{R}_1 \preceq \mathsf{R}_2$ and $\mathsf{R}_2 \preceq \mathsf{R}_1$ then we write $\mathsf{R}_1 \simeq \mathsf{R}_2$. If $\mathsf{R}_1 \preceq \mathsf{R}_2$ and not $\mathsf{R}_2 \preceq \mathsf{R}_1$, then $\mathsf{R}_1 \prec \mathsf{R}_2$.

The set of all $\preceq$-maximal elements in a set of sorts $S \subseteq \mathcal{R}$ is denoted $\max(S)$. R is a lower bound of S if $\mathsf{R} \preceq \mathsf{Q}$ for all $\mathsf{Q} \in S$. A lower bound G of S is the

greatest lower bound, denoted $\mathrm{glb}(S)$, if $\mathsf{R} \preceq \mathsf{G}$ for all lower bounds R of S. The sort $(\sum_{\mathsf{s}\in\mathcal{B}} \mathsf{s})^*$ is the *top sort* and is denoted by $\top$. Obviously, $\mathsf{R} \preceq \top$ for any R.

Terms. For each R we assume a countable set of variables $\mathcal{V}_\mathsf{R}$ such that $\mathcal{V}_{\mathsf{R}_1} = \mathcal{V}_{\mathsf{R}_2}$ iff $\mathsf{R}_1 \simeq \mathsf{R}_2$ and $\mathcal{V}_{\mathsf{R}_1} \cap \mathcal{V}_{\mathsf{R}_2} = \emptyset$ if $\mathsf{R}_1 \not\simeq \mathsf{R}_2$. Also, we assume as a signature a family of sets of function symbols $\{\mathcal{F}_{\mathsf{R}.\mathsf{s}} \mid \mathsf{R} \in \mathcal{R}, \mathsf{s} \in \mathcal{B}\}$ such that $\mathcal{F}_{\mathsf{R}_1.\mathsf{s}_1} = \mathcal{F}_{\mathsf{R}_2.\mathsf{s}_2}$ iff $\mathsf{R}_1.\mathsf{s}_1 \simeq \mathsf{R}_2.\mathsf{s}_2$. Moreover, the following conditions should be satisfied:

- Monotonicity: If $f \in \mathcal{F}_{\mathsf{R}_1.\mathsf{s}_1} \cap \mathcal{F}_{\mathsf{R}_2.\mathsf{s}_2}$ and $\mathsf{R}_1 \preceq \mathsf{R}_2$, then $\mathsf{s}_1 \preceq \mathsf{s}_2$.
- Preregularity: If $f \in \mathcal{F}_{\mathsf{R}_1.\mathsf{s}_1}$ and $\mathsf{R}_2 \preceq \mathsf{R}_1$, then there is a $\preceq$-least element in the set $\{\mathsf{s} \mid f \in \mathcal{F}_{\mathsf{R}.\mathsf{s}} \text{ and } \mathsf{R}_2 \preceq \mathsf{R}\}$.
- Finite overloading: For each f, the set $\{\mathcal{F}_{\mathsf{R}.\mathsf{s}} \mid f \in \mathcal{F}_{\mathsf{R}.\mathsf{s}}\}$ is finite.

We say that R is a sort of x if $x \in \mathcal{V}_\mathsf{R}$. Similarly, $\mathsf{R}.\mathsf{s}$ is a sort of f if $f \in \mathcal{F}_{\mathsf{R}.\mathsf{s}}$. Function symbols from $\mathcal{F}_{1.\mathsf{s}}$ are called constants. We use the letters a, b, c to denote them. $\mathit{maxsort}(f)$ denotes the set $\max(\{\mathsf{R}.\mathsf{s} \mid f \in \mathcal{F}_{\mathsf{R}.\mathsf{s}}\})$. We will write $f : \mathsf{R} \to \mathsf{s}$ for $f \in \mathcal{F}_{\mathsf{R}.\mathsf{s}}$, $a : \mathsf{s}$ for $a \in \mathcal{F}_{1.\mathsf{s}}$, and $x : \mathsf{R}$ for $x \in \mathcal{V}_\mathsf{R}$. Setting $\mathcal{V} \in \cup_{\mathsf{R}\in\mathcal{R}} \mathcal{V}_\mathsf{R}$ and $\mathcal{F} = \cup_{\mathsf{R}\in\mathcal{R},\mathsf{s}\in\mathcal{B}} \mathcal{F}_{\mathsf{R}.\mathsf{s}}$, we define the set of *sorted terms* (or, just *terms*) $\mathcal{T}(\mathcal{F}, \mathcal{V})$ over $\mathcal{F}$ and $\mathcal{V}$ as the least family $\{\mathcal{T}_\mathsf{R}(\mathcal{F}, \mathcal{V}) \mid \mathsf{R} \in \mathcal{R}\}$ of sets satisfying the following conditions:

- $\mathcal{V}_\mathsf{R} \subseteq \mathcal{T}_\mathsf{R}(\mathcal{F}, \mathcal{V})$.
- $\mathcal{T}_{\mathsf{R}_1}(\mathcal{F}, \mathcal{V}) \subseteq \mathcal{T}_{\mathsf{R}_2}(\mathcal{F}, \mathcal{V})$ if $\mathsf{R}_1 \preceq \mathsf{R}_2$.
- If $f : \mathsf{R} \to \mathsf{s}$ and $1 \preceq \mathsf{R}$, then $f(\epsilon) \in \mathcal{T}_\mathsf{s}(\mathcal{F}, \mathcal{V})$.
- If $f : \mathsf{R} \to \mathsf{s}$, $t_i \in \mathcal{T}_{\mathsf{R}_i}(\mathcal{F}, \mathcal{V})$ for $1 \le i \le n$, $n \ge 1$, such that $\mathsf{R}_1.\cdots.\mathsf{R}_n \preceq \mathsf{R}$, then $f(t_1, \ldots, t_n) \in \mathcal{T}_\mathsf{s}(\mathcal{F}, \mathcal{V})$.

We abbreviate terms $a(\epsilon)$ with a.

Lemma 1. *For each term t there exists a $\preceq$-minimal sort R that is unique modulo $\simeq$ such that $t \in \mathcal{T}_\mathsf{R}(\mathcal{F}, \mathcal{V})$.*

This $\preceq$-minimal sort R is called *the sort* of t and is denoted by $\mathit{sort}(t)$. In the same way, the sort of a term sequence $(t_1, \ldots, t_n)$, $n \ge 1$, is defined uniquely modulo $\simeq$ as $\mathit{sort}(t_1).\cdots.\mathit{sort}(t_n)$ and is denoted by $\mathit{sort}((t_1, \ldots, t_n))$. When $n = 0$, i.e., for the empty sequence, $\mathit{sort}(\epsilon) = 1$.

The set of variables of a term t is denoted by $\mathit{var}(t)$. A term t is ground if $\mathit{var}(t) = \emptyset$. These notions extend to term sequences, sets of term sequences, etc. For a basic sort s, its semantics $\mathit{sem}(\mathsf{s})$ is the set $\mathcal{T}_\mathsf{s}(\mathcal{F})$ of ground terms of sort s. Semantics of a regular sort is given as a set of ground term sequences of the corresponding sort: $\mathit{sem}(1) = \{\epsilon\}$, $\mathit{sem}(\mathsf{R}_1.\mathsf{R}_2) = \{(\tilde{s}_1, \tilde{s}_2) \mid \tilde{s}_1 \in \mathit{sem}(\mathsf{R}_1), \tilde{s}_2 \in \mathit{sem}(\mathsf{R}_2)\}$, $\mathit{sem}(\mathsf{R}_1{+}\mathsf{R}_2) = \mathit{sem}(\mathsf{R}_1) \cup \mathit{sem}(\mathsf{R}_2)$, $\mathit{sem}(\mathsf{R}^*) = \mathit{sem}(\mathsf{R})^*$. This definition, together with the definition of $\preceq$ and $\mathcal{T}_\mathsf{R}(\mathcal{F}, \mathcal{V})$ implies that if $\mathsf{R} \preceq \mathsf{Q}$, then $\mathit{sem}(\mathsf{R}) \subseteq \mathit{sem}(\mathsf{Q})$.

Substitutions. A substitution is a well-sorted mapping from variables to sequences of terms, which is identity almost everywhere. (A singleton sequence is identified with its sole member.) Substitutions are denoted with lower case

Greek letters, where ε stands for the identity substitution. Well-sortedness of σ means that $sort(\sigma(x)) \preceq sort(x)$ for all x. The notions of substitution application, term and term sequence instances, substitution composition, restriction, and subsumption are defined in the standard way. We use postfix notation for instances, juxtaposition for composition, and write $\sigma \leq_{\mathcal{X}} \vartheta$ for subsumption meaning that σ is more general than ϑ on the set of variables $\mathcal{X}$.

Lemma 2. *For a term t, a term sequence $\tilde{t}$, and a substitution σ we have $sort(t\sigma) \preceq sort(t)$ and $sort(\tilde{t}\sigma) \preceq sort(\tilde{t})$.*

Equation is a pair of term sequences, written as $\tilde{s} \doteq \tilde{t}$. A regular expression order sorted unification or, shortly, REOSU problem Γ is a finite set of equations between sorted term sequences $\{\tilde{s}_1 \doteq \tilde{t}_1, \ldots, \tilde{s}_n \doteq \tilde{t}_n\}$. A substitution σ is a unifier of Γ if $\tilde{s}_i\sigma = \tilde{t}_i\sigma$ for all $1 \leq i \leq n$. A minimal complete set of unifiers of Γ is a set U of unifiers of Γ satisfying the following conditions:

- Completeness: For any unifier ϑ of Γ there is $\sigma \in U$ such that $\sigma \leq_{var(\Gamma)} \vartheta$.
- Minimality: If there are $\sigma_1, \sigma_2 \in U$ such that $\sigma_1 \leq_{var(\Gamma)} \sigma_2$, then $\sigma_1 = \sigma_2$.

We also consider a *closure* $\overline{\mathsf{R}}$ of a sort R, defined as follows: $\overline{\mathsf{s}} = \sum_{\mathsf{r} \preceq \mathsf{s}} \mathsf{r}$, $\overline{1} = 1$, $\overline{\mathsf{R}_1.\mathsf{R}_2} = \overline{\mathsf{R}}_1.\overline{\mathsf{R}}_2$, $\overline{\mathsf{R}_1{+}\mathsf{R}_2} = \overline{\mathsf{R}}_1{+}\overline{\mathsf{R}}_2$, $\overline{\mathsf{R}^*} = \overline{\mathsf{R}}^*$. Closure makes operations on sorts considered later easier.

Lemma 3. *Let $\mathsf{S}, \mathsf{R} \in \mathcal{R}$. Then $\mathsf{S} \preceq \mathsf{R}$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$.*

Corollary 1. *Let $\mathsf{S}, \mathsf{R} \in \mathcal{R}$. Then $\mathsf{S} \simeq \mathsf{R}$ iff $[\![\overline{\mathsf{S}}]\!] = [\![\overline{\mathsf{R}}]\!]$.*

Linear Form and Split of a Regular Expression. We recall the notion of linear form for regular expressions from [2], adapted the notation to our setting, using the set of basic sorts $\mathcal{B}$ for alphabet: The pair (s, R) is called a monomial. A *linear form* of a regular expression R, denoted $lf(\mathsf{R})$, is a finite set of monomials defined recursively as follows:

$$\begin{array}{rlrl} lf(1) = \emptyset & & lf(\mathsf{R}^*) = lf(\mathsf{R}) \odot \mathsf{R}^* & \\ lf(\mathsf{s}) = \{(\mathsf{s}, 1)\} & & lf(\mathsf{R}.\mathsf{Q}) = lf(\mathsf{R}) \odot \mathsf{Q} & \text{if } \epsilon \notin [\![\mathsf{R}]\!] \\ lf(\mathsf{s}{+}\mathsf{r}) = lf(\mathsf{s}) \cup lf(\mathsf{r}) & & lf(\mathsf{R}.\mathsf{Q}) = lf(\mathsf{R}) \odot \mathsf{Q} \cup lf(\mathsf{Q}) & \text{if } \epsilon \in [\![\mathsf{R}]\!] \end{array}$$

These equations involve an extension of concatenation $\odot$ that acts on a linear form and a regular expression and returns a linear form. It is defined as $l \odot 1 = l$ and $l \odot \mathsf{Q} = \{(\mathsf{s}, \mathsf{S}.\mathsf{Q}) \mid (\mathsf{s}, \mathsf{S}) \in l, \mathsf{S} \neq 1\} \cup \{(\mathsf{s}, \mathsf{Q}) \mid (\mathsf{s}, 1) \in l\}$ if $\mathsf{Q} \neq 1$.

As an example, $lf(\mathsf{R}) = \{(\mathsf{s}, \mathsf{R}), (\mathsf{s}, \mathsf{s}.(\mathsf{s}{+}\mathsf{r})^*), (\mathsf{r}, (\mathsf{s}{+}\mathsf{r})^*)\}$ for $\mathsf{R} = \mathsf{s}^*.(\mathsf{s}.\mathsf{s}{+}\mathsf{r})^*$. The set $\overline{lf}(\mathsf{R})$ is defined as $\{\mathsf{s}.\mathsf{Q} \mid (\mathsf{s}, \mathsf{Q}) \in lf(\mathsf{R})\}$.

Definition 1 (Split). *Let $\mathsf{S} \in \mathcal{R}$. A* split *of S is a pair $(\mathsf{Q}, \mathsf{R}) \in \mathcal{R}^2$ such that (1) $\mathsf{Q}.\mathsf{R} \preceq \mathsf{S}$ and (2) if $(\mathsf{Q}', \mathsf{R}') \in \mathcal{R}^2$, $\mathsf{Q} \preceq \mathsf{Q}'$, $\mathsf{R} \preceq \mathsf{R}'$, and $\mathsf{Q}'.\mathsf{R}' \preceq \mathsf{S}$, then $\mathsf{Q} \simeq \mathsf{Q}'$ and $\mathsf{R} \simeq \mathsf{R}'$.*

We recall the definition of 2-factorization from [6]: A pair $(\mathsf{Q}, \mathsf{R}) \in \mathcal{R}^2$ is a *2-factorization* of $\mathsf{S} \in \mathcal{R}$ if (1) $[\![\mathsf{Q}.\mathsf{R}]\!] \subseteq [\![\mathsf{S}]\!]$ and (2) if $(\mathsf{Q}', \mathsf{R}') \in \mathcal{R}^2$, $[\![\mathsf{Q}]\!] \subseteq [\![\mathsf{Q}']\!]$, $[\![\mathsf{R}]\!] \subseteq [\![\mathsf{R}']\!]$, and $[\![\mathsf{Q}'.\mathsf{R}']\!] \subseteq [\![\mathsf{S}]\!]$, then $[\![\mathsf{Q}]\!] = [\![\mathsf{Q}']\!]$ and $[\![\mathsf{R}]\!] = [\![\mathsf{R}']\!]$.

Lemma 4. (Q, R) *is a split of* S *iff* $(\overline{\mathsf{Q}}, \overline{\mathsf{S}})$ *is a 2-factorization of* $\overline{\mathsf{S}}$.

In [6] it has been shown that regular expressions admit finite factorization: The number of left factor-right factor pairs is finite. Moreover, they can be effectively computed. By the lemma above regular expressions admit finite splitting. For instance, we can split $\mathsf{s}^*.\mathsf{r}.\mathsf{r}^*$ into $(\mathsf{s}^*, \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*)$ and $(\mathsf{s}^*.\mathsf{r}.\mathsf{r}^*, \mathsf{r}^*)$.

3 Algorithms for Sorts

Deciding $\preceq$. If we did not have ordering on basic sorts, $\preceq$ would be the standard inequality for regular word expressions which can be decided, for instance, by Antimirov's algorithm [1] that employs partial derivatives. The problem is PSPACE-complete, but this rewriting approach has an advantage over the standard technique of translating regular expressions into automata: With it, in some cases solving derivations can have polynomial size, while any algorithm based on translation of regular expressions into DFA's causes an exponential blow-up.

In our case, we can rely on the property $\mathsf{S} \preceq \mathsf{R}$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$, proved in Lemma 3. To decide the later inclusion, we do not need to take into account ordering on basic sorts. Hence, it can be decided by the original Antimirov's algorithm on $\overline{\mathsf{S}}$ and $\overline{\mathsf{R}}$.

Computing Greatest Lower Bounds. A greatest lower bound of regular expressions would be their intersection, if we did not have ordering on the basic sorts. Intersection can be computed either in the standard way, by translating them into automata, or by Antimirov & Mosses's rewriting algorithm [3] for regular expressions extended with the intersection operator. Computation requires double exponential time.

Here we can employ the regular expression intersection algorithm to compute a greatest lower bound, with one modification: To compute intersection between two alphabet letters, instead of standard check whether they are the same, we compute the maximal elements in the set of their lower bounds. There can be several such maximal elements. This can be easily computed based on the ordering on basic sorts. Then we can take the sum of these elements and it will be their greatest lower bound. This construction allows to compute a greatest lower bound of two regular expressions, which is unique modulo $\simeq$.

An implementation of Antimirov-Mosses algorithm (see [14]) requires only minor modification to deal with the ordering on alphabet letters (basic sorts). Hence, for S and R we compute here $\mathrm{glb}(\mathsf{S}, \mathsf{R})$ and we know that if Q is a regular expression with $[\![\mathsf{Q}]\!] = [\![\overline{\mathsf{S}}]\!] \cap [\![\overline{\mathsf{R}}]\!]$, then $\mathrm{glb}(\mathsf{S}, \mathsf{R}) \simeq \mathsf{Q}$.[3]

Computing Weakening Substitutions. Now we describe an algorithm that computes a substitution to weaken the sort of a term sequence towards a given sort. The necessity of such an algorithm can be demonstrated on the simple example:

[3] We say that the computation of glb fails, if the (modification of) Antimirov-Mosses algorithm returns 0, and express it $\mathrm{glb}(\mathsf{S}, \mathsf{R}) = \bot$.

Assume we want to unify x and $f(y)$ for $x : \mathsf{s}$, $f : \mathsf{R}_1 \to \mathsf{s}_1$, $f : \mathsf{R}_2 \to \mathsf{s}_2$, $y : \mathsf{R}_2$. Assume the sorts are ordered as $\mathsf{R}_1 \preceq \mathsf{R}_2$, $\mathsf{s}_1 \preceq \mathsf{s} \preceq \mathsf{s}_2$. Then we can not unify x with $f(y)$ directly, because $sort(f(y)) = \mathsf{s}_2 \not\preceq \mathsf{s} = sort(x)$. However, if we weaken the sort of $f(y)$ to s_1, then unification is possible. To weaken the sort of $f(y)$, we take its instance under the substitution $\{y \mapsto w\}$, where $sort(w) = \mathsf{R}_1$, which gives $sort(f(w)) = \mathsf{s}_1$. Hence, the substitution $\{y \mapsto w, x \mapsto f(w)\}$ is a unifier of x and $f(y)$, leading to the common instance $f(w)$.

A weakening pair is a pair of a term sequence $\tilde{t}$ and a sort Q, written $\tilde{t} \rightsquigarrow \mathsf{Q}$. A substitution ω is called a *weakening substitution* of a set W of weakening pairs iff $sort(\tilde{t}\omega) \preceq \mathsf{Q}$ for each $\tilde{t} \rightsquigarrow \mathsf{Q} \in W$.

The algorithm $\mathfrak{W}$ that is supposed to compute weakening substitutions for a given set of weakening pairs, works by applying exhaustively the following rules, selecting a weakening pair and transforming it in all possible ways:

R-w: **Remove a Weakening Pair**

$\{\tilde{t} \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow W; \sigma$ if $sort(\tilde{t}) \preceq \mathsf{Q}$.

D1-w: **Decomposition 1 in Weakening**

$\{(f(\tilde{t}), \tilde{s}) \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \{f(\tilde{t}) \rightsquigarrow \mathsf{s}, \tilde{s} \rightsquigarrow \mathsf{S}\} \cup W; \sigma$

if $sort((f(\tilde{t}), \tilde{s})) \not\preceq \mathsf{Q}$, $var((f(\tilde{t}), \tilde{s})) \neq \emptyset$, $\tilde{s} \neq \epsilon$ and $\mathsf{s}.\mathsf{S} \in \max(\overline{lf}(\mathsf{Q}))$.

D2-w: **Decomposition 2 in Weakening**

$\{(x, \tilde{s}) \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \{x \rightsquigarrow \mathsf{Q}_1, \tilde{s} \rightsquigarrow \mathsf{Q}_2\} \cup W; \sigma$

if $sort((x, \tilde{s})) \not\preceq \mathsf{Q}$, $\tilde{s} \neq \epsilon$ and $(\mathsf{Q}_1, \mathsf{Q}_2)$ is a split of Q.

AS-w: **Argument Sequence Weakening**

$\{f(\tilde{t}) \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \{\tilde{t} \rightsquigarrow \mathsf{R}\} \cup W; \sigma$

where $sort(f(\tilde{t})) \not\preceq \mathsf{Q}$, $var(f(\tilde{t})) \neq \emptyset$, $\mathsf{R}.\mathsf{r} \in maxsort(f)$, and $\mathsf{r} \preceq \mathsf{Q}$.

V-w: **Variable Weakening**

$\{x \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow W\sigma; \sigma\{x \mapsto w\}$

where w is a fresh variable with $sort(w) = \mathrm{glb}(\{sort(x), \mathsf{Q})\})$.

F1-w: **Failure 1 in Weakening**

$\{\tilde{t} \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \bot,$ if $sort(\tilde{t}) \not\preceq \mathsf{Q}$ and $var(\tilde{t}) = \emptyset$.

F2-w: **Failure 2 in Weakening**

$\{f(\tilde{t}) \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \bot,$

if $sort(f(\tilde{t})) \not\preceq \mathsf{Q}$ and $\mathsf{r} \not\preceq \mathsf{Q}$ for each r with $f : \mathsf{R} \to \mathsf{r}$.

F3-w: **Failure 3 in Weakening**

$\{x \rightsquigarrow \mathsf{Q}\} \cup W; \sigma \Longrightarrow \bot,$ if $\mathrm{glb}(\{sort(x), \mathsf{Q}\}) = \bot$.

By the exhaustive search described above, a complete search tree is generated whose branches form *derivations*. The branches that end with $\bot$ are called failing branches. The branches that end with $\emptyset; \omega$ are called successful branches and ω

is a substitution computed by $\mathfrak{W}$ at this branch. The set of all substitutions computed by $\mathfrak{W}$ on the set of weakening pairs W is denoted by $weak(W)$. It is easy to see that the elements of $weak(W)$ are variable renaming substitutions.

It is essential that the signature has the finite overloading property, which guarantees that the rule AS-w does not introduce infinite branching. Since the linear form and split of a regular expression are both finite, the other rules do not cause infinite branching either. $\mathfrak{W}$ is terminating, sound, and complete, as the following theorems show.

Theorem 1. *$\mathfrak{W}$ is terminating.*

Theorem 2. *$\mathfrak{W}$ is sound: Each $\omega \in weak(W)$ is a weakening substitution of W.*

Theorem 3. *$\mathfrak{W}$ is complete: For every weakening substitution ω' of W there exists $\omega \in weak(W)$ such that $\omega \leq_{vars(W)} \omega'$.*

Example 1. Let $W = \{x \rightsquigarrow \mathsf{q}, f(x) \rightsquigarrow \mathsf{s}\}$ be a weakening problem with $x : \mathsf{r}$, $f : \mathsf{s} \to \mathsf{s}$, $f : \mathsf{r} \to \mathsf{r}$ and the sorts $\mathsf{r}_1 \prec \mathsf{r}$, $\mathsf{r}_2 \prec \mathsf{r}$, $\mathsf{r}_1 \prec \mathsf{q}$, $\mathsf{r}_2 \prec \mathsf{q}$, $\mathsf{s} \prec \mathsf{r}_1$, $\mathsf{s} \prec \mathsf{r}_2$. Then the weakening algorithm computes $weak(W) = \{\{x \mapsto z, w \mapsto z\}\}$ where $w : \mathsf{r}_1{+}\mathsf{r}_2$ and $z : \mathsf{s}$.

Example 2. Let $W = \{(x, y) \rightsquigarrow \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*\}$ be a weakening problem with $x : \mathsf{q}_1^*.\mathsf{p}_1^*$, $y : \mathsf{q}_2^*.\mathsf{p}_2^*$, and the sorts $\mathsf{s} \prec \mathsf{q}_1$, $\mathsf{s} \prec \mathsf{q}_2$, $\mathsf{r} \prec \mathsf{p}_1$, $\mathsf{r} \prec \mathsf{p}_2$. Then the weakening algorithm computes $weak(W) = \{\{x \mapsto u_1, y \mapsto v_1\}, \{x \mapsto u_2, y \mapsto v_2\}\}$ where $u_1 : \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*$, $v_1 : \mathsf{r}^*$, $u_2 : \mathsf{s}^*$ and $v_2 : \mathsf{s}^*.\mathsf{r}.\mathsf{r}^*$.

Example 3. Let $W = \{x \rightsquigarrow \mathsf{q}^*\}$ be a weakening problem with $x : \mathsf{r}^*$ and the sorts $\mathsf{s}_1 \prec \mathsf{r}$, $\mathsf{s}_2 \prec \mathsf{r}$, $\mathsf{s}_1 \prec \mathsf{q}$, $\mathsf{s}_2 \prec \mathsf{q}$, $\mathsf{p}_1 \prec \mathsf{s}_1$, $\mathsf{p}_2 \prec \mathsf{s}_2$. Then the weakening algorithm computes $weak(W) = \{\{x \mapsto w\}\}$ where $w : (\mathsf{s}_1{+}\mathsf{s}_2)^*$.

4 Unification Type, Decidability and Unification Procedure

Unification Type. Let Γ_{re} be a REOSU problem and Γ_{seq} its version without sorts, i.e. a SEQU problem. Each unifier of Γ_{re} is either an unifier of Γ_{seq} or is obtained from an unifier of Γ_{seq} be composing it with a weakening substitution as follows: If $\sigma = \{x_1 \mapsto \tilde{t}_1, \ldots, x_n \mapsto \tilde{t}_n\}$ is a unifier of Γ_{seq}, then the set of weakening substitutions for σ is $\Omega(\sigma) = weak(\{\tilde{t}_1 \rightsquigarrow sort(x_1), \ldots, \tilde{t}_n \rightsquigarrow sort(x_n)\})$. For each $\omega_\sigma \in \Omega(\sigma)$, $\sigma\omega_\sigma$ is a unifier of Γ_{re}. Since SEQU is infinitary, the type of REOSU can be either infinitary or nullary, and we show now that it is not nullary.

Let S_{seq} be a minimal complete set of unifiers of Γ_{seq} and S_{re} be the set containing the unifiers of Γ_{re} that are either in S_{seq} or are obtained by weakening unifiers in S_{re}. Since $\{\sigma\omega_\sigma \mid \omega_\sigma \in \Omega(\sigma)\}$ is finite for each σ, we can assume that S_{re} contains only a minimal subset of it for each σ. The set S_{re} is complete. Assume by contradiction that it is not minimal. Then it contains σ' and ϑ' such that $\sigma' \leq_{var(\Gamma_{\mathrm{re}})} \vartheta'$, i.e., there exists φ' such that $\sigma'\varphi' =_{var(\Gamma_{\mathrm{re}})} \vartheta'$. If $\vartheta' \in S_{\mathrm{seq}}$,

then we have $\sigma'\varphi' = \sigma\omega_\sigma\varphi' =_{var(\Gamma)} \vartheta'$ for an $\omega_\sigma \in \Omega(\sigma)$, which contradicts minimality of S_{seq}. If $\sigma' \in S_{\text{seq}}$, then $\sigma'\varphi' =_{var(\Gamma_{\text{re}})} \vartheta' = \vartheta\omega_\vartheta$ where $\omega_\vartheta \in \Omega(\vartheta)$. Since ω_ϑ is variable renaming, $\sigma'\varphi'\omega_\vartheta^{-1} =_{var(\Gamma_{\text{seq}})} \vartheta$, which again contradicts minimality of S_{seq}. Both σ' and ϑ' can not be from S_{seq} because S_{seq} is minimal. If neither σ' nor ϑ' is in S_{seq}, then we have $\sigma\omega_\sigma\varphi' = \sigma'\varphi' =_{var(\Gamma_{\text{re}})} \vartheta' = \vartheta\omega_\vartheta$ and again a contradiction: $\sigma\omega_\sigma\varphi'\omega_\vartheta^{-1} =_{var(\Gamma_{\text{seq}})} \vartheta$.

Hence, for any Γ_{re} there is a complete set of unifiers with no two elements comparable with respect to $\leq_{var(\Gamma_{\text{re}})}$, which implies that Γ_{re} has a minimal complete set of unifiers and REOSU is not nullary.

Is REOSU Decidable? In this subsection we reduce solvability of REOSU to solvability of a unification problem in the union of two disjoint theories: word unification with regular constraints and order-sorted syntactic unification with finitely many sorts. Sort symbols can be shared. Each of these problems is decidable [13, 16], but we do not know whether it implies decidability of REOSU. A version with regular constraints (or an order-sorted version) of Baader-Schulz combination method [4] would imply it, if the restrictions on the ingredient theories remain as reasonable as for the unsorted case (e.g. linear constant restrictions). However, we are not aware of any work on such a combination method.

Here we describe solvability-preserving transformations that transform a given REOSU problem Γ_1 into a problem of the union theory. First, we use variable abstraction on Γ_1, replacing each subterm at depth 1 or deeper with a fresh variable of the same sort. For instance, with this transformation from $\Gamma = \{x \doteq f(g(y), a), g(x, f(z)) \doteq g(f(z), x)\}$ we obtain a new REOSU problem $\Gamma_2 = \{x \doteq f(u, v), u \doteq g(y), v \doteq a, g(x, w) \doteq g(w, x), w \doteq f(z)\}$.

In Γ_2 there can be equations of the form $x \doteq t$ such that $sort(t) \not\preceq sort(x)$. If t is ground, we can immediately stop with failure. Otherwise, we weaken t towards $sort(x)$, apply weakening substitutions on Γ_2 obtaining finitely many instances, and repeat this step on each instance until we reach unification problems $\Gamma_3^1, \ldots, \Gamma_3^k$ such that each equation of the form $x \doteq t$ in them satisfies $sort(t) \preceq sort(x)$. The problems where it is not possible are discarded as unsolvable. Γ_1 is solvable iff Γ_3^i is solvable for some $1 \leq i \leq k$.

Now, Γ_3's are transformed as follows: First, we introduce a new least sort n and a new function symbol $conc : \top \to \mathsf{n}$. Next, we introduce new function symbols $f' : \mathsf{n} \to \mathsf{s}$ for each $f : \mathsf{R} \to \mathsf{s}$. The obtained signature is denoted by $\mathcal{F}'$. Then, every term in Γ_3 of the form $f(x_1, \ldots, x_n)$ is replaced by $f'(conc(x_1, \ldots, x_n))$. The new problem is denoted with Γ_4. This transformation preserves solvability: If σ is a solution of Γ_3, then Γ_4 has a solution σ', obtained from σ with the same transformation as Γ_4 was obtained from Γ_3. On the other hand, if σ' solves Γ_4, we can obtain σ from it by getting rid of $conc$ and replacing each f' with f. The obtained mapping is well-sorted and, therefore, is a substitution. It solves Γ_3.

Γ_4 is a unification problem over the union of two disjoint signatures: $\mathcal{F}'$ and $\{conc\}$. The signature $\mathcal{F}'$ contains unary function symbols only, whose domain sort is n. We have only basic sorts (not regular expression sorts) in this theory, and they are finitely many. The theory with the signature $conc$ can be seen as

elementary word unification problem with regular expression sorts. Hence, Γ_4 is a unification problem in the combination of two disjoint theories that share basic sorts: One is a first-order order-sorted unification with finitely many sorts and the other one is word unification with regular expression sorts. Decidability of the first one follows for decidability of elementary order-sorted unification [16]. Decidability of the second one follows from decidability of word unification with regular constraints, because one can encode the sort information as regular constraints over an extended alphabet. What is left to show is that the Baader-Schulz method remains correct in the presence of regular constraints (or in the order-sorted case). We conjecture this is true, but do not have a formal proof at the moment of writing this paper.

Conjecture 1. REOSU is decidable.

Unification Procedure. To compute unifiers for a REOSU problem, we could employ the SEQU unification procedure and then weaken each computed substitution. However, this is a generate and test approach that can be made better if we tailor weakening in the unification rules. This is what we do now.

The rules for REOSU procedure below transform a pair of a unification problem and a substitution into either again such a pair.

P: **Projection**

$\Gamma; \sigma \Longrightarrow \Gamma\vartheta; \sigma\vartheta,$

for $\vartheta = \{x_1 \mapsto \epsilon, \ldots, x_n \mapsto \epsilon\}$ with $x_i \in var(\Gamma)$ and $1 \preceq sort(x_i)$ for $1 \leq i \leq n$.

T: **Trivial**

$\{\tilde{t} \doteq \tilde{t}\} \cup \Gamma; \sigma \Longrightarrow \Gamma; \sigma.$

TP: **Trivial Prefix**

$\{(\tilde{r}, \tilde{t}) \doteq (\tilde{r}, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq \tilde{s}\} \cup \Gamma; \sigma, \qquad$ if $\tilde{r} \neq \epsilon$ and $\tilde{t} \neq \tilde{s}$.

D: **Decomposition**

$\{(f(\tilde{t}), \tilde{t}') \doteq (f(\tilde{s}), \tilde{s}')\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq \tilde{s}, \tilde{t}' \doteq \tilde{s}'\} \cup \Gamma; \sigma,$

if $\mathrm{glb}(\{sort(f(\tilde{t})), sort(f(\tilde{s}))\}) \neq \bot$ and $\tilde{t} \neq \tilde{s}$.

O: **Orient**

$\{(t, \tilde{t}) \doteq (x, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{(x, \tilde{s}) \doteq (t, \tilde{t})\} \cup \Gamma; \sigma, \qquad$ where $t \notin \mathcal{V}$.

WkE1: **Weakening and Elimination 1**

$\{(x, \tilde{t}) \doteq (s, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

where $s \notin \mathcal{V}$, $x \notin var(s)$, $\omega \in weak(\{s \rightsquigarrow sort(x)\})$, and $\vartheta = \omega \cup \{x \mapsto s\omega\}$.

WkE2: **Weakening and Elimination 2**

$\{(x, \tilde{t}) \doteq (y, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{\tilde{t} \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

where $\vartheta = \{x \mapsto w, y \mapsto w\}$ for a fresh variable w whose sort $sort(w) = \mathrm{glb}(\{sort(x), sort(y)\})$ and $sort(w) \not\preceq 1$.

WkWd1: **Weakening and Widening 1**

$\{(x, \tilde{t}) \doteq (s, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{(z, \tilde{t}) \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

if $s \notin \mathcal{V}$, $x \notin var(s)$, there is $(\mathsf{r}, \mathsf{R}) \in lf(sort(x))$ with $\mathsf{R} \not\geq 1$, $\omega \in weak(\{s \rightsquigarrow \mathsf{r}\})$, z is a fresh variable with $sort(z) = \mathsf{R}$ and $\vartheta = \omega \cup \{x \mapsto (s\omega, z)\}$.

WkWd2: **Weakening and Widening 2**

$\{(x, \tilde{t}) \doteq (y, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{(z, \tilde{t}) \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

where (S, R) is a split of $sort(x)$ such that $\mathsf{R} \not\geq 1$, w is a fresh variable with $sort(w) = \mathrm{glb}(\{\mathsf{S}, sort(y)\})$ and $sort(w) \not\geq 1$, z is a fresh variable with $sort(z) = \mathsf{R}$, and $\vartheta = \{x \mapsto (w, z),\ y \mapsto w\}$.

WkWd3: **Weakening and Widening 3**

$\{(x, \tilde{t}) \doteq (y, \tilde{s})\} \cup \Gamma; \sigma \Longrightarrow \{(z, \tilde{t}) \doteq \tilde{s}\}\vartheta \cup \Gamma\vartheta; \sigma\vartheta,$

where (S, R) is a split of $sort(y)$ such that $\mathsf{R} \not\geq 1$, w is a fresh variable with $sort(w) = \mathrm{glb}(\{\mathsf{S}, sort(x)\})$ and $sort(w) \not\geq 1$, z is a fresh variable with $sort(z) = \mathsf{R}$, and $\vartheta = \{x \mapsto w,\ y \mapsto (w, z)\}$.

Note that $sort(w) \not\geq 1$ in WkWd2 and WkWd3 implies that in those rules $\mathsf{S} \not\geq 1$.

We denote this set of transformation rules with $\mathfrak{T}$.

Theorem 4 (Soundness). *The rules in $\mathfrak{T}$ are sound.*

To solve an unification problem Γ, we create the initial pair $\Gamma; \varepsilon$ and first apply the projection rule to it in all possible ways. From each obtained problem we select and equation and apply the other rules exhaustively to that selected equation, developing the search tree in a breadth-first way. If no rule applies, the problem is transformed $\bot$. The obtained procedure is denoted by $\mathfrak{P}(\Gamma)$. Branches in the search tree form *derivations*. The derivations that end with $\bot$ are *failing derivations*. The derivations that end with $\emptyset; \varphi$ are *successful derivations*. The set of all φ's at the end of successful derivations of $\mathfrak{P}(\Gamma)$ is called the *computed substitution set* of $\mathfrak{P}(\Gamma)$ and is denoted by $comp(\mathfrak{P}(\Gamma))$. From Theorem 4 by induction on the length of derivations one can prove that every $\varphi \in comp(\mathfrak{P}(\Gamma))$ is a unifier of Γ.

One can observe that under this control, variables are replaced with ϵ only at the projection phase. In particular, no variable introduced in intermediate stages gets eliminated with ϵ or replaced by a variable whose sort is 1.

Theorem 5 (Completeness). *Let Γ be a REOSU problem with a unifier ϑ. Then there exists $\sigma \in comp(\mathfrak{P}(\Gamma))$ such that $\sigma \leq_{var(\Gamma)} \vartheta$.*

Note that the set $comp(\mathfrak{P}(\Gamma))$, in general, is not minimal.[4]

[4] However, if in the rules WkE1 and WkE2 the substitution ω is selected from a minimal subset of the corresponding weakening set, one can show that $comp(\mathfrak{P}(\Gamma))$ is almost minimal. (Almost minimality is defined in [10]). We do not go into the details here.

5 Conclusion

We studied unification in order-sorted theories with regular expression sorts. We showed how it generalizes some known unification problems, conjectured its decidability and gave a complete unification procedure. As the language can model quite naturally DTD and in some extent, XML Schema, one can see a possible application (perhaps of its fragments) in the area related to XML processing.

References

1. V. Antimirov. Rewriting regular inequalities (extended abstract). In H. Reichel, editor, *Fundamentals of Computation Theory, 10th International Symposium FCT'95*, volume 965 of *LNCS*, pages 116–125. Springer, 1995.
2. V. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.
3. V. Antimirov and P. D. Mosses. Rewriting extended regular expressions. *Theor. Comput. Sci.*, 143(1):51–72, 1995.
4. F. Baader and K. U. Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *J. Symbolic Computation*, 21(2):211–244, 1996.
5. F. Baader and W. Snyder. Unification theory. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 8, pages 445–532. Elsevier Science, 2001.
6. J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.
7. J. A. Goguen and J. Meseguer. Order-sorted algebra i: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theor. Comput. Sci.*, 105(2):217–273, 1992.
8. J. Hendrix and J. Meseguer. Order-sorted unification revisited. In G. Kniesel and J. Sousa Pinto, editors, *Pre-proceedings of the 9th International Workshop on Rule-Based Programming, RULE'09*, pages 16–29, 2008.
9. C. Kirchner. Order-sorted equational unification. Presented at the fifth International Conference on Logic Programming (Seattle, USA), 1988. Also as rapport de recherche INRIA 954, Dec. 88.
10. T. Kutsia. Solving equations with sequence variables and sequence functions. *J. Symbolic Computation*, 42(3):352–388, 2007.
11. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.
12. M. Schmidt-Schauß. *Computational Aspects of an Order-sorted Logic with Term Declarations*. Number 395 in Lecture Notes in Computer Science. Springer, 1989.
13. K. U. Schulz. Makanin's algorithm for word equations – two improvements and a generalization. In K. Schulz, editor, *Word Equations and Related Topics*, number 572 in LNCS, pages 85–150. Springer, 1990.
14. M. Sulzmann. regexpr-symbolic: Regular expressions via symbolic manipulation. http://hackage.haskell.org/cgi-bin/hackage-scripts/package/regexpr-symbolic, 2009.
15. Ch. Walther. Many-sorted unification. *J. ACM*, 35(1):1–17, 1988.
16. Ch. Weidenbach. Unification in sort theories and its applications. *Annals of Mathematics and Artificial Intelligence*, 18(2):261–293, 1996.

A Proofs of Lemmas and Theorems

Lemma 1. *For each term t there exists a $\preceq$-minimal sort R that is unique modulo $\simeq$ such that $t \in \mathcal{T}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$.*

Proof. By structural induction. Let t be a variable $x \in \mathcal{V}_{\mathsf{R}}$. Assume $x \in \mathcal{T}_{\mathsf{Q}}(\mathcal{F}, \mathcal{V})$ for some Q. It implies that $x \in \mathcal{V}_{\mathsf{Q}'}$ with $\mathsf{Q}' \preceq \mathsf{Q}$. Then $x \in \mathcal{V}_{\mathsf{R}} \cap \mathcal{V}_{\mathsf{Q}'}$, which implies $\mathsf{R} \simeq \mathsf{Q}'$. Hence, R is the unique $\preceq$-minimal sort modulo $\simeq$ with $x \in \mathcal{T}_{\mathsf{R}}(\mathcal{F}, \mathcal{V})$.

If t is a constant, the lemma follows from the monotonicity condition.

If $t = f(t_1, \ldots, t_n)$, then, by the induction hypothesis, each t_i has the $\preceq$-minimal sort R_i, $1 \leq i \leq n$, that is unique modulo $\simeq$. Let $f : \mathsf{Q} \to \mathsf{q}$ such that $\mathsf{R}_1 . \cdots . \mathsf{R}_n \preceq \mathsf{Q}$. By preregularity, there exists a $\preceq$-least s such that $f : \mathsf{S} \to \mathsf{s}$ and $\mathsf{R}_1 . \cdots . \mathsf{R}_n \preceq \mathsf{S}$. Hence, s is the $\preceq$-minimal sort such that $t \in \mathcal{T}_{\mathsf{s}}(\mathcal{F}, \mathcal{V})$. □

Lemma 2. *For a term t, a term sequence $\tilde{t}$, and a substitution σ we have $sort(t\sigma) \preceq sort(t)$ and $sort(\tilde{t}\sigma) \preceq sort(\tilde{t})$.*

Proof. We prove $sort(t\sigma) \preceq sort(t)$ by induction on term structure. If t is a variable, then the lemma follows from the definition of substitution. If $t = f(\epsilon)$ then it is obvious. If $t = f(t_1, \ldots, t_n)$, $n \geq 1$, then by the induction hypothesis $sort(t_i\sigma) \preceq sort(t_i)$ for each $1 \leq i \leq n$. Therefore, $sort((t_1, \ldots, t_n)\sigma) \preceq sort(t_1, \ldots, t_n)$ which, by the definition of sorted terms, implies that $sort(t\sigma) = sort(f(t_1, \ldots, t_n)\sigma) \preceq sort(f(t_1, \ldots, t_n)) = sort(t)$.

To prove $sort(\tilde{t}\sigma) \preceq sort(\tilde{t})$, we use induction on the length of $\tilde{t}$. If the length is 0, then the lemma is obvious. Otherwise, let $\tilde{t}$ be $(t, \tilde{t}')$. By the case above we have $sort(t\sigma) \preceq sort(t)$. By the induction hypothesis we get $sort(\tilde{t}'\sigma) \preceq sort(\tilde{t}')$. Therefore, $sort(\tilde{t}\sigma) = sort((t, \tilde{t}')\sigma) \preceq sort((t, \tilde{t}')) = sort(\tilde{t})$. □

Lemma 3. *Let $\mathsf{S}, \mathsf{R} \in \mathcal{R}$. Then $\mathsf{S} \preceq \mathsf{R}$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$.*

Proof. For any $\mathsf{Q} \in \mathcal{R}$ and $v \in [\![\mathsf{Q}]\!]$, let $v{\downarrow}\mathsf{Q} := \{w \in [\![\mathsf{Q}]\!] \mid w \preceq v\}$. By the definition of closure, we have that $\{\max(v{\downarrow}\overline{\mathsf{Q}}) \mid v \in [\![\overline{\mathsf{Q}}]\!]\} = [\![\mathsf{Q}]\!]$. Now, we can reason as follows: $\mathsf{S} \preceq \mathsf{R}$ iff $[\![\mathsf{S}]\!] \preceq [\![\mathsf{R}]\!]$ iff $\{\max(v{\downarrow}\overline{\mathsf{S}}) \mid v \in [\![\overline{\mathsf{S}}]\!]\} \preceq \{\max(v{\downarrow}\overline{\mathsf{R}}) \mid v \in [\![\overline{\mathsf{R}}]\!]\}$ iff $[\![\overline{\mathsf{S}}]\!] \subseteq [\![\overline{\mathsf{R}}]\!]$. □

Lemma 4. *(Q, R) is a split of S iff $(\overline{\mathsf{Q}}, \overline{\mathsf{S}})$ is a 2-factorization of $\overline{\mathsf{S}}$.*

Proof. (Q, R) is a split of S iff (1) $\mathsf{Q}.\mathsf{R} \preceq \mathsf{S}$ and (2) if $(\mathsf{Q}', \mathsf{R}') \in \mathcal{R}^2$, $\mathsf{Q} \preceq \mathsf{Q}'$, $\mathsf{R} \preceq \mathsf{R}'$, and $\mathsf{Q}'.\mathsf{R}' \preceq \mathsf{S}$, then $\mathsf{Q} \simeq \mathsf{Q}'$ and $\mathsf{R} \simeq \mathsf{R}'$. By Lemma 3, these conditions are equivalent to (1') $[\![\overline{\mathsf{Q}.\mathsf{R}}]\!] \subseteq [\![\overline{\mathsf{S}}]\!]$ and (2') if $(\mathsf{Q}', \mathsf{R}') \in \mathcal{R}^2$, $[\![\overline{\mathsf{Q}}]\!] \subseteq [\![\overline{\mathsf{Q}'}]\!]$, $[\![\overline{\mathsf{R}}]\!] \subseteq [\![\overline{\mathsf{R}'}]\!]$, and $[\![\overline{\mathsf{Q}'.\mathsf{R}'}]\!] \subseteq [\![\overline{\mathsf{S}}]\!]$, then $[\![\overline{\mathsf{Q}}]\!] = [\![\overline{\mathsf{Q}'}]\!]$ and $[\![\overline{\mathsf{R}}]\!] = [\![\overline{\mathsf{R}'}]\!]$. It is not hard to see that (1') and (2') are the same as saying that $(\overline{\mathsf{Q}}, \overline{\mathsf{S}})$ is a 2-factorization of $\overline{\mathsf{S}}$. □

Theorem 1. *The algorithm $\mathfrak{W}$ is terminating.*

Proof. The complexity measure of a weakening pair $\tilde{t} \rightsquigarrow \mathsf{Q}$ is 1 + the denotational length of $\tilde{t}$, and the complexity measure of a set W of weakening pairs is the multiset of the complexity measures of its constituent weakening pairs.

The multiset extension of the standard ordering on nonnegative integers is well-founded. The first five rules in $\mathfrak{W}$ strictly decrease the measure for the sets they operate on, and the failure rules cause immediate termination. Hence, $\mathfrak{W}$ is terminating. □

Theorem 2. *The algorithm $\mathfrak{W}$ is sound: Every $\omega \in weak(W)$ is a weakening substitution of W.*

Proof. It is enough to show that if a rule in $\mathfrak{W}$ transforms $W_1;\sigma$ into $W_2;\sigma\vartheta$ and φ is a weakening substitution for W_2, then $\vartheta\varphi$ is a weakening substitution for W_1. For R-w, Lemma 2 implies it. For D1-w it follows from two facts: First, if $\mathsf{s.S} \in \max(\overline{lf}(\mathsf{Q}))$ then $\mathsf{s.S} \preceq \mathsf{Q}$, and second, $\preceq$-monotonicity of concatenation: If $\mathsf{R}_1 \preceq \mathsf{Q}_1$ and $\mathsf{R}_2 \preceq \mathsf{Q}_2$ then $\mathsf{R}_1.\mathsf{R}_2 \preceq \mathsf{Q}_1.\mathsf{Q}_2$. For D1-w it follows from $\preceq$-monotonicity of concatenation and from the definition of split. For AS-w, it is implied by the definition of *maxsort*, for V-w by the definition of glb and Lemma 2. □

Theorem 3. *The algorithm $\mathfrak{W}$ is complete: For every weakening substitution ω' of W there there exists $\omega \in weak(W)$ such that $\omega \leq_{vars(W)} \omega'$.*

Proof. We construct the desired derivation recursively. The initial pair is $W;\varepsilon$. Assume that $W_i;\sigma_i$ belongs to the derivation. Then there exists φ such that $\sigma_i\varphi =_{var(W)} \omega'$. Moreover, φ is a weakening substitution of $W\sigma_i$ and W_i. We want to extend the derivation with $W_{i+1};\sigma_{i+1}$ such that $\sigma_{i+1} \leq_{var(W)} \omega'$. Let $\tilde{r} \rightsquigarrow \mathsf{Q}$ be the selected weakening pair in W_i. If $sort(\tilde{r}) \preceq \mathsf{Q}$, we proceed with R-w. If $sort(\tilde{r}) \not\preceq \mathsf{Q}$, we have several cases depending on the shape of $\tilde{r}$:

- $\tilde{r} = (f(\tilde{t}), \tilde{s})$, $\tilde{s} \neq \epsilon$. $\tilde{r}$ is not ground, otherwise φ would not be its weakening substitution. We select $\mathsf{s.S} \in \max(\overline{lf}(\mathsf{Q}))$ such that $sort(f(\tilde{t})\varphi) \preceq \mathsf{s}$ and $sort(\tilde{s}\varphi) \preceq \mathsf{Q}$ and proceed with D1-w. Then φ is a weakening substitution of W_{i+1} and $\sigma_{i+1} = \sigma_i \leq_{var(W)} \omega'$.
- $\tilde{r} = (x, \tilde{s})$, $\tilde{s} \neq \epsilon$. We select a split $(\mathsf{Q}_1, \mathsf{Q}_2)$ of Q such that $sort(x\varphi) \preceq \mathsf{Q}_1$ and $sort(\tilde{s}\varphi) \preceq \mathsf{Q}_2$ and continue with D2-w. Again, φ is a weakening substitution of W_{i+1} and $\sigma_{i+1} = \sigma_i \leq_{var(W)} \omega'$.
- $\tilde{r} = f(\tilde{t})$. $\tilde{r}$ is not ground, otherwise φ would not be its weakening substitution. We select R and s such that $\mathsf{R.r} \in maxsort(f)$, $\mathsf{r} \preceq \mathsf{Q}$, and $sort(\tilde{t}\varphi) \preceq \mathsf{R}$. Such R and r exist, because φ weakens W_i. Then we continue with the rule AS-w. φ is a weakening substitution of W_{i+1} and $\sigma_{i+1} = \sigma_i \leq_{var(W)} \omega'$.
- $\tilde{r} = x$. Then there exists w with $sort(w) = \mathrm{glb}(sort(x), \mathsf{Q})$ such that $\varphi = \{x \mapsto w\}\varphi'$. We select such a w and continue derivation with V-w and the substitution $\vartheta = \{x \mapsto w\}$. Then φ' is a weakening substitution of W_{i+1} and $\sigma_{i+1} = \sigma_i\vartheta \leq_{var(W)} \omega'$.

Hence, we constructed the desired extension in all the cases. Since the algorithm is terminating, the derivation reaches a success end $\emptyset;\omega$ with the property $\omega \leq_{vars(W)} \omega'$. □

Theorem 4 (Soundness). *The rules in $\mathfrak{T}$ are sound.*

Proof. It is easy to to check for each rule in $\mathfrak{T}$ that if it performs a transformation $\Gamma, \sigma \Longrightarrow \Delta, \sigma\vartheta$ and φ is a unifier of Δ, then $\vartheta\varphi$ is a unifier of Γ. □

Theorem 5 (Completeness). *Let Γ be a REOSU problem with a unifier ϑ. Then there exists $\sigma \in comp(\mathfrak{P}(\Gamma))$ such that $\sigma \leq_{var(\Gamma)} \vartheta$.*

Proof. We construct recursively the derivation that computes σ. The initial pair in the derivation is $\Gamma; \varepsilon$ and $\varepsilon \leq_{var(\Gamma)} \vartheta$. To choose a proper extension, we find all $x \in var(\Gamma)$ with $x\vartheta = \epsilon$ and make the projection step with the substitution σ_1 whose domain consists of these x's only. Obviously, $\sigma_1 \leq_{var(\Gamma)} \vartheta$.

Now assume $\Gamma_n; \sigma_n$ belongs to the derivation. Then $\sigma_n \leq_{var(\Gamma)} \vartheta$, i.e., there exists φ such that $x\sigma_n\varphi = x\vartheta$ for all $x \in var(\Gamma)$. Moreover, it is easy to see that φ is a unifier of both $\Gamma\sigma_n$ and Γ_n and $x\varphi \neq \epsilon$ for any x. We want to extend the derivation with $\Gamma_{n+1}, \sigma_{n+1}$ such that $\sigma_{n+1} \leq_{var(\Gamma)} \vartheta$. Let $\tilde{t} \doteq \tilde{s}$ be the selected equation in Γ_n and represent Γ_n as $\{\tilde{t} \doteq \tilde{s}\} \cup \Gamma'_n$. Then we have the following cases:

1. $\tilde{t}$ and $\tilde{s}$ are either identical, or have identical nonempty prefixes, or their first elements are nonvariable terms with the same head. Then $\Gamma_{n+1}; \sigma_{n+1}$ is obtained by the rules T, TP, or D, respectively. Hence, $\sigma_{n+1} = \sigma_n \leq_{var(\Gamma)} \vartheta$.
2. The first element of $\tilde{s}$ is a variable x, while $\tilde{t}$ does not start with a variable. Then we apply the rule O and get $\sigma_{n+1} = \sigma_n \leq_{var(\Gamma)} \vartheta$.
3. The first element of $\tilde{t}$ is a variable x, while $\tilde{s}$ does not start with a variable. Since φ is a unifier of Γ_n and does not map x to ϵ, we have either $x\varphi = s\varphi$ or $x\varphi = (s\varphi, \tilde{s}')$ where s is the first element of $\tilde{s}$ and $\tilde{s}' \neq \epsilon$. In the first case, $sort(s\varphi) \preceq sort(x)$, i.e., φ involves weakening of $sort(s)$ to $sort(x)$. We single out this weakening substitution ω from φ and extend the derivation with the rule $\mathsf{WkE1}$ and substitution $\omega \cup \{x \mapsto s\omega\}$. In the second case we have $sort(s\varphi).sort(\tilde{s}') \preceq sort(x)$. Since $sort(s)$ is basic sort, there exists $(\mathsf{r}, \mathsf{R}) \in lf(\mathsf{R})$ such that $sort(s\varphi) \preceq \mathsf{r}$ and $sort(\tilde{s}') \preceq \mathsf{R}$. Hence, φ involves weakening of of $sort(s)$ to r. Therefore, extracting the weakening substitution ω from φ, we extend the derivation by $\mathsf{WkWd1}$ and $\omega \cup \{x \mapsto (s\omega, z)\}$, where z is fresh with $sort(z) = \mathsf{R}$. In either case $\sigma_{n+1} \leq_{var(\Gamma)} \vartheta$.
4. The first element of $\tilde{t}$ is a variable x and the first element of $\tilde{t}$ is another variable y. There are the following alternatives: $x\varphi = y\varphi$, $x\varphi = (y\varphi, \tilde{s}')$, or $y\varphi = (x\varphi, \tilde{t}')$, where $\tilde{s}' \neq \epsilon$ and $\tilde{t}' \neq \epsilon$. In the first case, φ also involves weakening for x and y and we proceed with $\mathsf{WkE2}$ with the corresponding weakening substitution. In the second case $sort(y\varphi) \preceq \mathsf{S}$ and $sort(\tilde{s}') \preceq \mathsf{R}$ for a split (S, R) of $sort(x)$. We choose such a split and proceed with the rule $\mathsf{WkWd2}$ together with a properly chosen substitution $\{x \mapsto (w, z), y \mapsto w\}$. The third case is analogous to the second one. In all the cases we have $\sigma_{n+1} \leq_{var(\Gamma)} \vartheta$.

The second step is to show that this sequence terminates. We define inductively the size of a term t, sequence of terms $\tilde{t}$, and a substitution σ with respect to a substitution ϑ as follows: $|x|_\vartheta = 0$, if $x\vartheta = \epsilon$, otherwise $|x|_\vartheta = 1$. $|f(\tilde{t})|_\vartheta = |\tilde{t}|_\vartheta + 2$, $|(t_1, \ldots, t_n)|_\vartheta = |t_1|_\vartheta + \cdots + |t_n|_\vartheta$, and $|\sigma|_\vartheta = \sum_{x \in dom(\sigma)} |x\sigma|_\vartheta$, where $dom(\sigma)$

is the domain of σ. Given Γ and ϑ, we define the size of $\Gamma_i; \sigma_i$ as the quadruple $|\Gamma_i; \sigma_i| = (k, l, m, n)$ where

- k is the number of distinct variables in Γ_i;
- $l = |\vartheta|_\varepsilon - |\sigma_i|_{var(\Gamma)}|_\vartheta$, where $\sigma_i|_{var(\Gamma)}$ is the restriction of σ_i on the set of variables on Γ;
- m is the multiset $\cup_{\tilde{t} \doteq \tilde{s} \in \Gamma_i} \{|\tilde{t}|_\vartheta, |\tilde{s}|_\vartheta\}$;
- n is the number of equations of the form $(t, \tilde{t}) \doteq (x, \tilde{s})$, $t \notin \mathcal{V}$ in Γ_i.

The sizes are compared lexicographically. The ordering is well-founded.

The projection rule is applied only once and does not increase the size of the unification problem/substitution pairs it operates on. The other rules strictly decrease the size: T, TP, D decrease m and do not increase k and l. O decreases n without increasing the others. WkE1 and WkE2 decrease k. WkWd1, WkWd2, and WkWd3 decrease l and do not increase k. Hence, the derivation we have constructed above terminates. □